1. Outliers

Banglore House Prices

Detecting and removing outliers

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
house_price = pd.read_csv("house_price.csv")
```

In [3]:
```
house_price.head(5)
```

Out[3]:

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250

In [4]:
```
# Display the data types, non-null counts, and memory usage.
house_price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13200 entries, 0 to 13199
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   location        13200 non-null  object
 1   size            13200 non-null  object
 2   total_sqft      13200 non-null  float64
 3   bath            13200 non-null  float64
 4   price           13200 non-null  float64
 5   bhk             13200 non-null  int64
 6   price_per_sqft  13200 non-null  int64
dtypes: float64(3), int64(2), object(2)
memory usage: 722.0+ KB
```

In [5]:
```
#Statistical summary
house_price.describe()
```

Out[5]:

	total_sqft	bath	price	bhk	price_per_sqft
count	13200.000000	13200.000000	13200.000000	13200.000000	1.320000e+04
mean	1555.302783	2.691136	112.276178	2.800833	7.920337e+03
std	1237.323445	1.338915	149.175995	1.292843	1.067272e+05
min	1.000000	1.000000	8.000000	1.000000	2.670000e+02
25%	1100.000000	2.000000	50.000000	2.000000	4.267000e+03
50%	1275.000000	2.000000	71.850000	3.000000	5.438000e+03
75%	1672.000000	3.000000	120.000000	3.000000	7.317000e+03

1. Mean method

In [9]:
```
# Calculate the mean and standard deviation of price per square feet
mean_price_per_sqft = house_price['price_per_sqft'].mean()
std_price_per_sqft = house_price['price_per_sqft'].std()
```

In [10]:
```
# Define a threshold for outliers (e.g., 3 standard deviations away from the mean)
threshold = 3 * std_price_per_sqft
```

In [11]:
```
# Detect outliers
outliers = house_price[(house_price['price_per_sqft'] < mean_price_per_sqft - threshold) |
                (house_price['price_per_sqft'] > mean_price_per_sqft + threshold)]
```

In [12]:
```
# Remove outliers using mean function
cleaned_data = house_price[(house_price['price_per_sqft'] >= mean_price_per_sqft - threshold) &
                    (house_price['price_per_sqft'] <= mean_price_per_sqft + threshold)]
```

In [13]:
```
# Analyze the dataset after removing outliers
print("Original dataset shape:", house_price.shape)
print("Cleaned dataset shape:", cleaned_data.shape)
```

```
Original dataset shape: (13200, 7)
Cleaned dataset shape: (13195, 7)
```

2. IQR

2. IQR

In [14]:
```
# Calculate the 25th and 75th percentiles of price per square feet
Q1 = house_price['price_per_sqft'].quantile(0.25)
Q3 = house_price['price_per_sqft'].quantile(0.75)
```

In [15]:
```
# Calculate the interquartile range (IQR)
IQR = Q3 - Q1
```

In [16]:
```
# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

In [17]:
```
# Detect outliers
outliers = house_price[(house_price['price_per_sqft'] < lower_bound) | (house_price['price_per_sqft'] > upper_bound)]
```

In [18]:
```
# Remove outliers using percentile method
cleaned_house_price = house_price[(house_price['price_per_sqft'] >= lower_bound) & (house_price['price_per_sqft'] <= upper_bound)
```

In [19]:
```
# Analyze the dataset after removing outliers
print("Original dataset shape:", house_price.shape)
print("Cleaned dataset shape:", cleaned_house_price.shape)
```

```
Original dataset shape: (13200, 7)
Cleaned dataset shape: (11935, 7)
```

3. Percentile Method

In [20]:
```
# Calculate the 5th and 95th percentiles of price per square feet
percentile_5 = house_price['price_per_sqft'].quantile(0.05)
percentile_95 = house_price['price_per_sqft'].quantile(0.95)
```

In [21]:
```
# Define the lower and upper bounds for outliers
lower_bound = percentile_5
upper_bound = percentile_95
```

In [22]:
```
# Detect outliers
outliers = house_price[(house_price['price_per_sqft'] < lower_bound) | (house_price['price_per_sqft'] > upper_bound)]
```

In [23]:
```
# Remove outliers using percentile method
cleaned_house_price = house_price[(house_price['price_per_sqft'] >= lower_bound) & (house_price['price_per_sqft'] <= upper_bound)
```

In [24]:
```
# Analyze the dataset after removing outliers
print("Original dataset shape:", house_price.shape)
print("Cleaned dataset shape:", cleaned_house_price.shape)
```

```
Original dataset shape: (13200, 7)
Cleaned dataset shape: (11880, 7)
```

4. Z-score method for normal distribution

[25]:
```
# Calculate mean and standard deviation of price per square feet
mean_price_per_sqft = house_price['price_per_sqft'].mean()
std_price_per_sqft = house_price['price_per_sqft'].std()
```

[26]:
```
# Calculate Z-score for each data point
house_price['z_score'] = (house_price['price_per_sqft'] - mean_price_per_sqft) / std_price_per_sqft
```

[27]:
```
# Define a threshold for outlier detection (e.g., Z-score greater than 3 or less than -3)
threshold = 3
```

[28]:
```
# Detect outliers
outliers = house_price[(house_price['z_score'] > threshold) | (house_price['z_score'] < -threshold)]
```

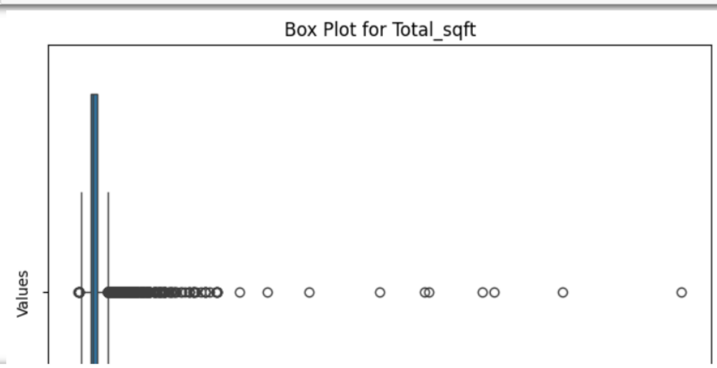
[29]:
```
# Remove outliers using Z-score method
cleaned_house_price = house_price[(house_price['z_score'] <= threshold) & (house_price['z_score'] >= -threshold)]
```

[30]:
```
# Analyze the dataset after removing outliers
print("Original dataset shape:", house_price.shape)
print("Cleaned dataset shape:", cleaned_house_price.shape)
```

```
Original dataset shape: (13200, 8)
Cleaned dataset shape: (13195, 8)
```

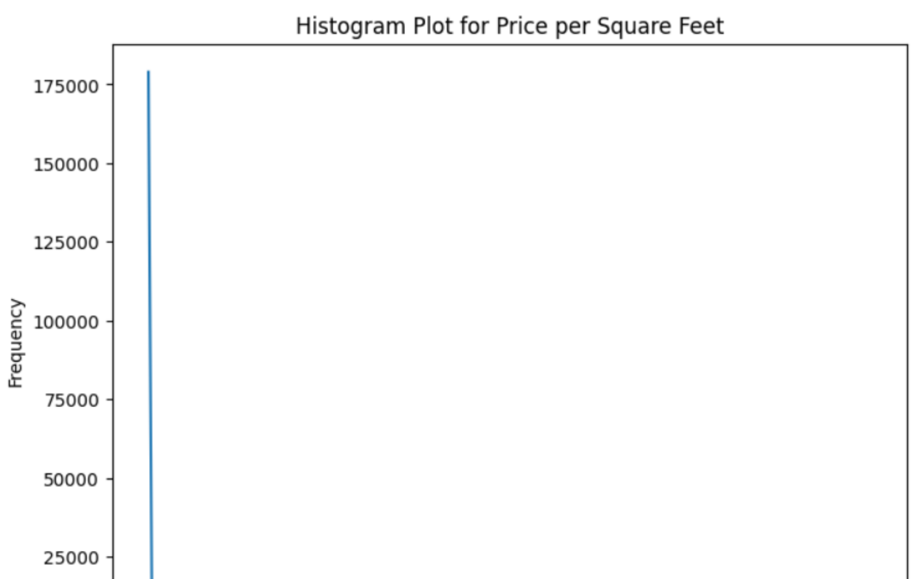
In [31]:

```python
# List of numerical columns
numerical_columns = ['total_sqft', 'bath', 'price', 'bhk', 'price_per_sqft']

# Plot a separate box plot for each numerical column
for column in numerical_columns:
    plt.figure(figsize=(8, 6))  # Set the figure size for each plot
    sns.boxplot(x=house_price[column])
    plt.title(f'Box Plot for {column.capitalize()}')
    plt.xlabel(column.capitalize())
    plt.ylabel('Values')
    plt.show()
```

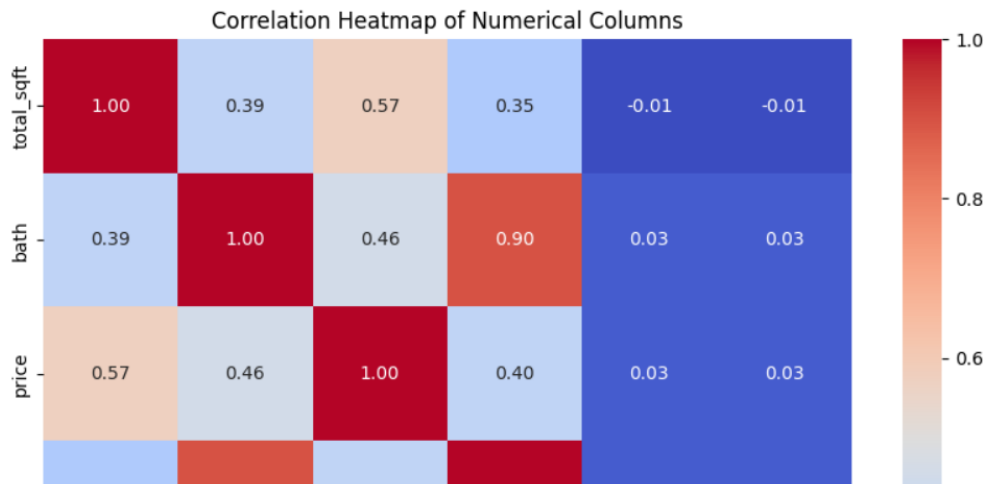


In [32]:

```python
# Plot a histogram to check the normality of the "price_per_sqft" column with adjusted bins
plt.figure(figsize=(8, 6))  # Set the figure size
sns.histplot(data=house_price['price_per_sqft'], bins=20, kde=True)  # Adjust the number of bins
plt.title('Histogram Plot for Price per Square Feet')
plt.xlabel('Price per Square Feet')
plt.ylabel('Frequency')
plt.show()
```



In [33]:
```
# Select only numerical columns for correlation calculation
numerical_columns = house_price.select_dtypes(include=['number'])

# Compute the correlation matrix
correlation_matrix = numerical_columns.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



2. Hypothesis testing

Q1.Suppose a child psychologist claims that the average time working mothers spend talking to their children is at least 11 minutes per day. You conduct a random sample of 1000 working mothers and find they spend an average of 11.5 minutes per day talking with their children. Assume prior research suggests the population standard deviation is 2.3 minutes.Conduct a test with a level of significance of alpha = 0.05.

In [1]:
```
import numpy as np
from scipy.stats import norm
```

In [2]:
```
# Set up the problem
sample_mean = 11.5
population_mean = 11
population_stddev = 2.3
sample_size = 1000
alpha = 0.05
```

In [3]:
```
# Calculate the test statistic (z-score)
z_score = (sample_mean - population_mean) / (population_stddev / np.sqrt(sample_size))
```

In [4]:
```

# Determine critical value
critical_value = norm.ppf(1 - alpha)
```

In [5]:
```
# Make decision
if z_score > critical_value:
    print("Reject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")
```

```
Reject the null hypothesis
```

test at a significance level of 0.05 and determine whether there is enough evidence to support the coffee shop's claim.

In [6]:
```python
from scipy.stats import t
```

In [7]:
```python
# Set up the problem
sample_mean = 4.6
population_mean = 5
sample_stddev = 0.8
sample_size = 40
alpha = 0.05
```

In [8]:
```python
# Calculate the test statistic (t-score)
t_score = (sample_mean - population_mean) / (sample_stddev / np.sqrt(sample_size))
```

In [9]:
```python
# Determine critical value
df = sample_size - 1  # degrees of freedom
critical_value = t.ppf(alpha, df)
```

In [10]:
```python

# Determine p-value
p_value = t.cdf(t_score, df)

# Make decision
if t_score < critical_value:
    print("Reject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")

print("t-score:", t_score)
print("Critical value:", critical_value)
print("p-value:", p_value)
```

```
Reject the null hypothesis
```

3. Data Preprocessing

jupyter 3. Data Preprocessing Last Checkpoint: Yesterday at 08:15 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

Employee Data

The main objective of this project is to design and implement a robust data preprocessing system that addresses common challenges such as missing values, outliers, inconsistent formatting, and noise. By performing effective data preprocessing, the project aims to enhance the quality, reliability, and usefulness of the data for machine learning.

Data Dictionary

Company- The name of the organization. Age: Age of the employee. Salary: Salary of the employee.Place: The place where the company is located. Country: The country where the company is located. Gender: 0 for male and 1 for female.

In [1]:
```python
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
employee = pd.read_csv("employee.csv")
```

Data Exploration and Cleaning

In [3]:
```
# Display the first 5 rowsa
employee.head()
```

Out[3]:

	Company	Age	Salary	Place	Country	Gender
0	TCS	20.0	NaN	Chennai	India	0
1	Infosys	30.0	NaN	Mumbai	India	0
2	TCS	35.0	2300.0	Calcutta	India	0
3	Infosys	40.0	3000.0	Delhi	India	0
4	TCS	23.0	4000.0	Mumbai	India	0

In [4]:
```
# Display the data types, non-null counts, and memory usage.
employee.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148 entries, 0 to 147
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Company  140 non-null    object 
 1   Age      130 non-null    float64
 2   Salary   124 non-null    float64
 3   Place    134 non-null    object 
 4   Country  148 non-null    object 
 5   Gender   148 non-null    int64  
dtypes: float64(2), int64(1), object(3)
memory usage: 7.1+ KB
```

In [5]:
```
#Statistical summary
employee.describe()
```

Out[5]:

	Age	Salary	Gender
count	130.000000	124.000000	148.000000
mean	30.484615	5312.467742	0.222973
std	11.096640	2573.764683	0.417654
min	0.000000	1089.000000	0.000000
25%	22.000000	3030.000000	0.000000
50%	32.500000	5000.000000	0.000000
75%	37.750000	8000.000000	0.000000
max	54.000000	9876.000000	1.000000

In [7]:
```
employee.shape
```

Out[7]:
```
(148, 6)
```

In [8]:
```
# Checking the null values
employee.isnull().sum()
```

Out[8]:
```
Company     8
Age        18
Salary     24
Place      14
Country     0
Gender      0
dtype: int64
```

We can observe that there are 8 null values in company, 18 in Age, 24 in Salary and 14 in Place.Let's correct that.

In [9]:
```python
employee["Company"].fillna(pd.NA, inplace=True)
employee["Age"].fillna(pd.NA, inplace=True)
employee["Salary"].fillna(pd.NA, inplace=True)
employee["Place"].fillna(pd.NA, inplace=True)
```

In [10]:
```python
employee.head(5)
```

Out[10]:

	Company	Age	Salary	Place	Country	Gender
0	TCS	20.0	NaN	Chennai	India	0
1	Infosys	30.0	NaN	Mumbai	India	0
2	TCS	35.0	2300.0	Calcutta	India	0
3	Infosys	40.0	3000.0	Delhi	India	0
4	TCS	23.0	4000.0	Mumbai	India	0

In [11]:
```python
duplicate_rows = employee[employee.duplicated()]

print("Duplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
     Company   Age  Salary     Place Country  Gender
84       CTS  43.0     NaN    Mumbai   India       0
130      TCS  21.0  4824.0    Mumbai   India       0
131  Infosys   NaN  5835.0    Mumbai   India       0
144  Infosys  22.0  8787.0  Calcutta   India       1
```

The dataset does not contain duplicates for the serial number. Therefore there is no reason to remove any rows.

Finding the unique values in each category

In [12]:
```python
unique_company = employee["Company"].unique()
unique_age = employee["Age"].unique()
unique_salary = employee["Salary"].unique()
unique_place = employee["Place"].unique()
unique_country = employee["Country"].unique()
unique_gender = employee["Gender"].unique()
```

In [13]:
```python
# Length of unique values
len_company = len(unique_company)
len_age = len(unique_age)
len_salary = len(unique_salary)
len_place = len(unique_place)
len_country = len(unique_country)
len_gender = len(unique_gender)
```

```python
print("Number of unique values in Company:", len_company)

print("Unique values in Age:", unique_age)
print("Number of unique values in Age:", len_age)

print("Unique values in Salary:", unique_salary)
print("Number of unique values in Salary:", len_salary)

print("Unique values in Place:", unique_place)
print("Number of unique values in Place:", len_place)

print("Unique values in Country:", unique_country)
print("Number of unique values in Country:", len_country)

print("Unique values in Gender:", unique_gender)
print("Number of unique values in Gender:", len_gender)
```

```
Unique values in Company: ['TCS' 'Infosys' 'CTS' <NA> 'Tata Consultancy Services' 'Congnizant'
 'Infosys Pvt Lmt']
Number of unique values in Company: 7
Unique values in Age: [20. 30. 35. 40. 23. nan 34. 45. 18. 22. 32. 37. 50. 21. 46. 36. 26. 41.
 24. 25. 43. 19. 38. 51. 31. 44. 33. 17.  0. 54.]
Number of unique values in Age: 30
Unique values in Salary: [  nan 2300. 3000. 4000. 5000. 6000. 7000. 8000. 9000. 1089. 1234. 3030.
 3045. 3184. 4824. 5835. 7084. 8943. 8345. 9284. 9876. 2034. 7654. 2934.
 4034. 5034. 8202. 9024. 4345. 6544. 6543. 3234. 4324. 5435. 5555. 8787.
 3454. 5654. 5009. 5098. 3033.]
Number of unique values in Salary: 41
Unique values in Place: ['Chennai' 'Mumbai' 'Calcutta' 'Delhi' 'Podicherry' 'Cochin' <NA> 'Noida'
 'Hyderabad' 'Bhopal' 'Nagpur' 'Pune']
Number of unique values in Place: 12
Unique values in Country: ['India']
Number of unique values in Country: 1
Unique values in Gender: [0 1]
Number of unique values in Gender: 2
```

In [15]:
```
# Replacce 0 in 'Age' with 'NaN'
employee["Age"].replace(0, pd.NA, inplace=True)
```

In [16]:
```
# Detecting the outliers in Age and Salary using IQR
# Calculate the IQR for Age and Salary
Q1_age = employee['Age'].quantile(0.25)
Q3_age = employee['Age'].quantile(0.75)
IQR_age = Q3_age - Q1_age

Q1_salary = employee['Salary'].quantile(0.25)
Q3_salary = employee['Salary'].quantile(0.75)
IQR_salary = Q3_salary - Q1_salary
```

In [17]:
```

# Define the upper and lower bounds
lower_bound_age = Q1_age - 1.5 * IQR_age
upper_bound_age = Q3_age + 1.5 * IQR_age

lower_bound_salary = Q1_salary - 1.5 * IQR_salary
upper_bound_salary = Q3_salary + 1.5 * IQR_salary
```

In [18]:
```

# Filter out outliers
employee_filtered = employee[
    (employee['Age'] >= lower_bound_age) & (employee['Age'] <= upper_bound_age) &
    (employee['Salary'] >= lower_bound_salary) & (employee['Salary'] <= upper_bound_salary)
]
```

In [19]:
```
# Mean Age and Mean Salary
import pandas as pd

# Assuming 'employee' is your DataFrame
mean_age = employee['Age'].mean()
mean_salary = employee['Salary'].mean()

print("Mean Age:", mean_age)
print("Mean Salary:", mean_salary)
```

```
Mean Age: 31.95967741935484
Mean Salary: 5312.467741935484
```

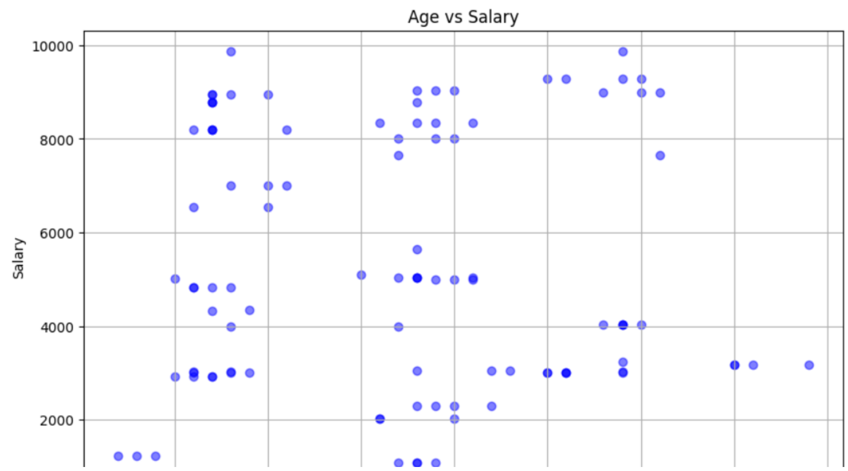
Data Analysis

In [20]:
```
# Employees of Age>40 and Salary<5000
filtered_data = employee[(employee['Age'] > 40) & (employee['Salary'] < 5000)]

# Print the filtered data
print(filtered_data)
```

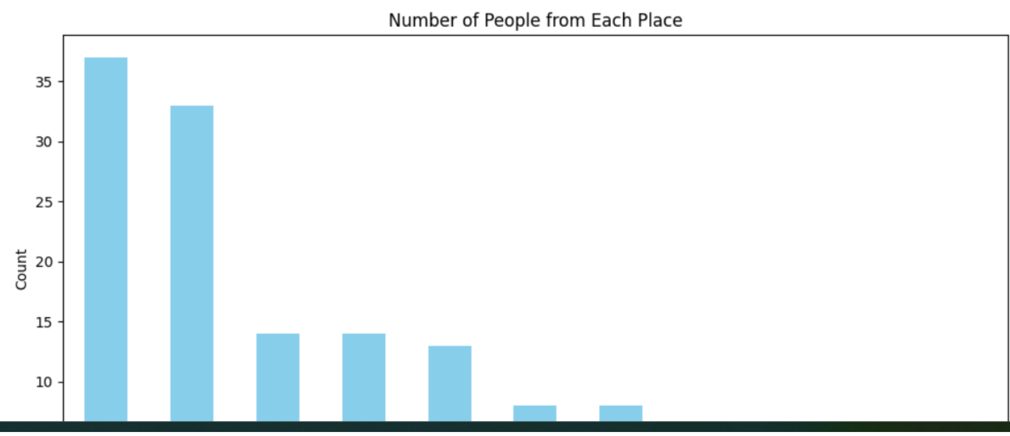
```
     Company   Age  Salary      Place Country  Gender
21   Infosys  50.0  3184.0      Delhi   India       0
32   Infosys  45.0  4034.0   Calcutta   India       0
39   Infosys  41.0  3000.0     Mumbai   India       0
50   Infosys  41.0  3000.0    Chennai   India       0
57   Infosys  51.0  3184.0  Hyderabad   India       0
68   Infosys  43.0  4034.0     Mumbai   India       0
75   Infosys  44.0  3000.0     Cochin   India       0
86   Infosys  41.0  3000.0      Delhi   India       0
93   Infosys  54.0  3184.0     Mumbai   India       0
104  Infosys  44.0  4034.0      Delhi   India       0
122  Infosys  44.0  3234.0     Mumbai   India       0
129  Infosys  50.0  3184.0   Calcutta   India       0
138      CTS  44.0  3033.0     Cochin   India       0
140  Infosys  44.0  4034.0  Hyderabad   India       0
145  Infosys  44.0  4034.0      Delhi   India       1
```

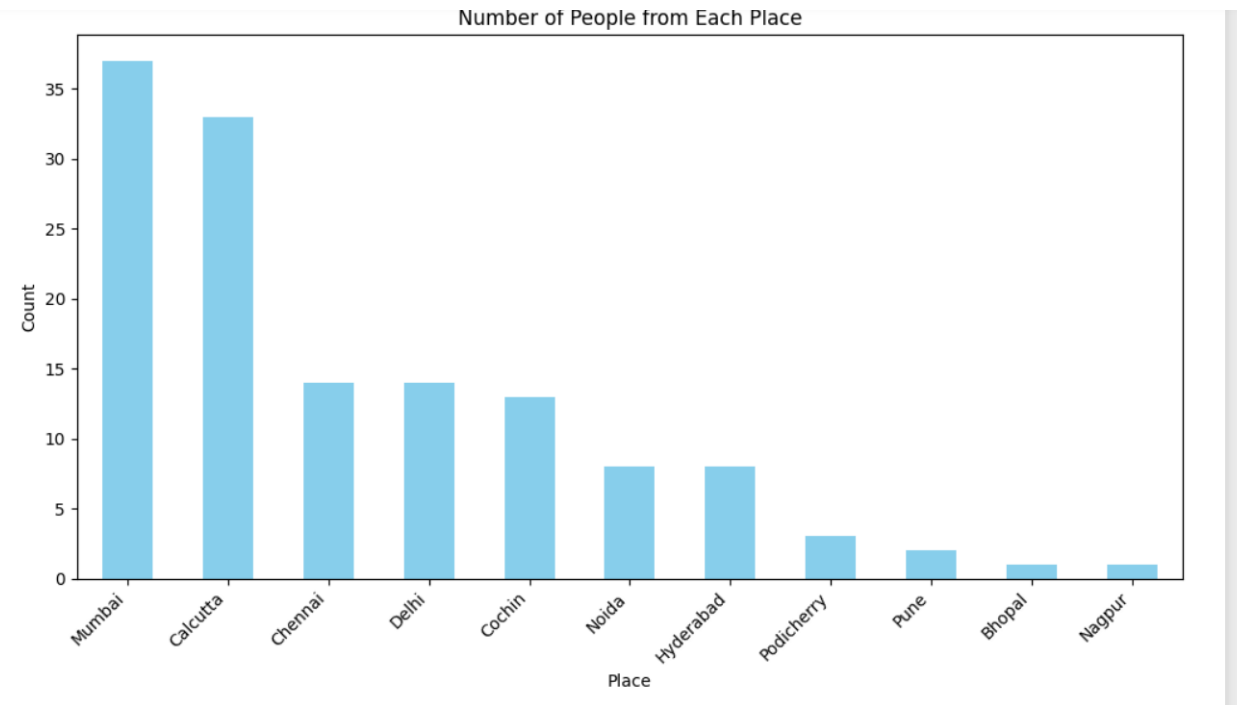
In [21]:
```
#Plot the  chart with age and salary

# Replace pd.NA with NaN
employee['Age'] = employee['Age'].fillna(np.nan)
employee['Salary'] = employee['Salary'].fillna(np.nan)

# Plot the chart
plt.figure(figsize=(10, 6))
plt.scatter(employee['Age'], employee['Salary'], color='blue', alpha=0.5)
```

```
plt.xlabel('Age')
plt.ylabel('Salary')
plt.grid(True)
plt.show()
```




```
In [22]: # Count the number of people from each place and represent it visually
place_counts = employee['Place'].value_counts()

# Plot the bar chart
plt.figure(figsize=(10, 6))
place_counts.plot(kind='bar', color='skyblue')
plt.title('Number of People from Each Place')
plt.xlabel('Place')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```





In [24]:
```python
# Label Encoding
from sklearn.preprocessing import LabelEncoder
# Convert all values to strings
employee['Company'] = employee['Company'].astype(str)
employee['Place'] = employee['Place'].astype(str)
employee['Country'] = employee['Country'].astype(str)

# Fill missing values with a placeholder or a common category
employee['Company'].fillna('Unknown', inplace=True)
employee['Place'].fillna('Unknown', inplace=True)
employee['Country'].fillna('Unknown', inplace=True)

# Perform label encoding
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
employee['Company'] = label_encoder.fit_transform(employee['Company'])
employee['Place'] = label_encoder.fit_transform(employee['Place'])
employee['Country'] = label_encoder.fit_transform(employee['Country'])
```

In [25]:
```python
print(employee)
```

```
     Company   Age  Salary  Place  Country  Gender
0          5  20.0     NaN      3        0       0
1          3  30.0     NaN      7        0       0
2          5  35.0  2300.0      2        0       0
3          3  40.0  3000.0      5        0       0
4          5  23.0  4000.0      7        0       0
..       ...   ...     ...    ...      ...     ...
143        5  33.0  9024.0      2        0       1
144        3  22.0  8787.0      2        0       1
145        3  44.0  4034.0      5        0       1
146        5  33.0  5034.0      7        0       1
```

Feature Scaling

In [26]:
```
# To determone which scaler to choose, find if the data falls under Gaussian distribution. For that Skewness and Kurtosis are use
skewness_age = employee['Age'].skew()
skewness_salary = employee['Salary'].skew()
kurtosis_age = employee['Age'].kurtosis()
kurtosis_salary = employee['Salary'].kurtosis()
print("Skewness of Age:", skewness_age)
print("Skewness of Salary:", skewness_salary)
print("Kurtosis of Age:", kurtosis_age)
print("Kurtosis of Salary:", kurtosis_salary)
```

```
Skewness of Age: 0.26712459210333117
Skewness of Salary: 0.1696390042066502
Kurtosis of Age: -0.9985725557794884
Kurtosis of Salary: -1.2555750103839947
```

While the skewness values suggest approximate symmetry, the negative kurtosis values suggest that both distributions are less peaked and have lighter tails than a normal distribution. Based on these measures, the distributions of Age and Salary may not perfectly fit a Gaussian distribution. Shapiro-Wilk Test: Use the Shapiro-Wilk test to assess the normality of Age and Salary. The null hypothesis is that the data is drawn from a normal distribution. If the p-value is greater than a chosen significance level (e.g., 0.05), you can't reject the null hypothesis, indicating that the data may be normally distributed.

In [27]:
```
from scipy.stats import shapiro

# Drop missing values
age_data = employee['Age'].dropna()
salary_data = employee['Salary'].dropna()

# Ensure numeric format
age_data = pd.to_numeric(age_data)
salary_data = pd.to_numeric(salary_data)

# Perform Shapiro-Wilk test
shapiro_age = shapiro(age_data)
shapiro_salary = shapiro(salary_data)

print("Shapiro-Wilk test for Age:", shapiro_age)
print("Shapiro-Wilk test for Salary:", shapiro_salary)
```

```
Shapiro-Wilk test for Age: ShapiroResult(statistic=0.9320949996074708, pvalue=9.55441908075672e-06)
Shapiro-Wilk test for Salary: ShapiroResult(statistic=0.9298811105610535, pvalue=6.8986807309056e-06)
```

Since both p-values are significantly smaller than the typical significance level of 0.05, we reject the null hypothesis that the data is normally distributed. Therefore we apply the min-max scaler.

n [28]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler_minmax = MinMaxScaler()

# Perform feature scaling using MinMaxScaler
scaled_features_minmax = scaler_minmax.fit_transform(employee)
```

n [29]:
```
print(employee)
```

```
     Company   Age  Salary  Place  Country  Gender
0          5  20.0     NaN      3        0       0
1          3  30.0     NaN      7        0       0
2          5  35.0  2300.0      2        0       0
3          3  40.0  3000.0      5        0       0
4          5  23.0  4000.0      7        0       0
..       ...   ...     ...    ...      ...     ...
143        5  33.0  9024.0      2        0       1
144        3  22.0  8787.0      2        0       1
```

4. Regression Testing

In [2]:
```python
car_price = pd.read_csv("CarPrice_Assignment.csv")
```

Data Exploration and Cleaning

In [3]:
```python
car_price.head(5)
```

Out[3]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

In [4]:
```python
# Display the data types, non-null counts, and memory usage.
car_price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [5]:
```python
#Statistical summary
car_price.describe()
```

Out[5]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepowe
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.00000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.142537	104.11707
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.00000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.00000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.00000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.00000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.00000

In [6]:
```python
car_price.shape
```

Out[6]: (205, 26)

In [7]:
```
# Checking the null values
car_price.isnull().sum()
```

Out[7]:
```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

There are no null values in any columns.

In [8]:
```
car_price.head(5)
```

Out[8]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

In [9]:
```
duplicate_rows = car_price[car_price.duplicated()]

print("Duplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
Empty DataFrame
Columns: [car_ID, symboling, CarName, fueltype, aspiration, doornumber, carbody, drivewheel, enginelocation, wheelbase, carleng
th, carwidth, carheight, curbweight, enginetype, cylindernumber, enginesize, fuelsystem, boreratio, stroke, compressionratio, h
orsepower, peakrpm, citympg, highwaympg, price]
Index: []

[0 rows x 26 columns]
```

There are no duplicate rows.

In [11]:
```python
# Check the length of unique values in each column
unique_lengths = car_price.nunique()
print("Length of unique values in each column:")
print(unique_lengths)
```

```
Length of unique values in each column:
car_ID              205
symboling             6
CarName             147
fueltype              2
aspiration            2
doornumber            2
carbody               5
drivewheel            3
enginelocation        2
wheelbase            53
carlength            75
carwidth             44
carheight            49
curbweight          171
enginetype            7
cylindernumber        7
enginesize           44
fuelsystem            8
boreratio            38
stroke               37
compressionratio     32
horsepower           59
peakrpm              23
citympg              29
highwaympg           30
price               189
dtype: int64
```

Data Preprocessing ¶

In [12]:
```python
# Drop the 'car_ID' column
car_price.drop('car_ID', axis=1, inplace=True)
```

In [13]:
```python
# Extract company name from 'CarName' column
car_price['company'] = car_price['CarName'].apply(lambda x: x.split()[0])
```

In [14]:
```python
# View unique company names after extraction
unique_companies = car_price['company'].unique()
print("Unique company names:")
print(unique_companies)
```

```
Unique company names:
['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'maxda' 'mazda' 'buick' 'mercury' 'mitsubishi' 'Nissan' 'nissan'
 'peugeot' 'plymouth' 'porsche' 'porcshce' 'renault' 'saab' 'subaru'
 'toyota' 'toyouta' 'vokswagen' 'volkswagen' 'vw' 'volvo']
```

In [15]:
```python
# Dictionary to map correct company names
corrections = {
    'maxda': 'mazda',
    'Nissan': 'nissan',
    'porcshce': 'porsche',
    'toyouta': 'toyota',
    'vokswagen': 'volkswagen',
    'vw': 'volkswagen'
}

# Correct spelling errors in company names
car_price['company'] = car_price['company'].replace(corrections)

# View unique company names after corrections
unique_companies_corrected = car_price['company'].unique()
print("Unique company names after corrections:")
print(unique_companies_corrected)
```

```
Unique company names after corrections:
['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'mazda' 'buick' 'mercury' 'mitsubishi' 'nissan' 'peugeot' 'plymouth'
 'porsche' 'renault' 'saab' 'subaru' 'toyota' 'volkswagen' 'volvo']
```

In [16]:
```python

# Perform label encoding for categorical columns
label_encoder = LabelEncoder()
categorical_cols = ['fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'enginetype', 'cylindernu
for col in categorical_cols:
    car_price[col] = label_encoder.fit_transform(car_price[col])
```

In [17]:
```python
# Function to detect and remove outliers using IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers_removed = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return outliers_removed

# Columns where outliers need to be detected and removed
columns_with_outliers = ['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'c

# Remove outliers for each column
for col in columns_with_outliers:
    car_price = remove_outliers(car_price, col)

# Check the shape of the DataFrame after removing outliers
print("Shape of DataFrame after removing outliers:", car_price.shape)
```

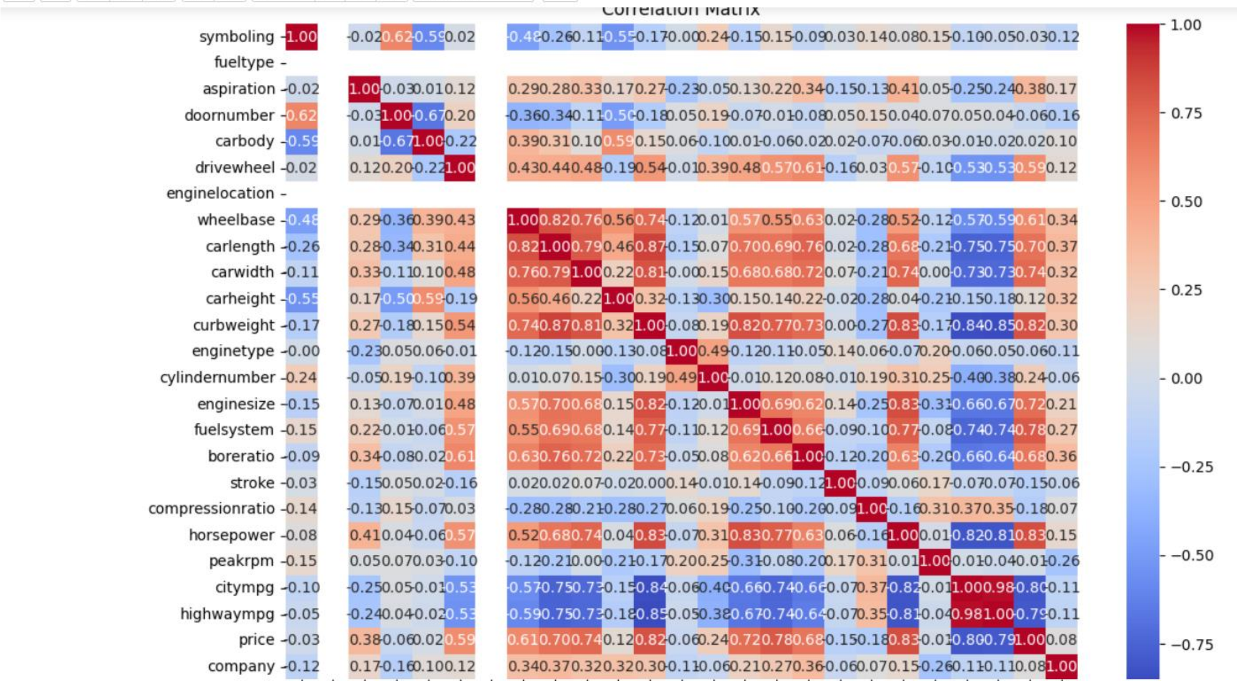
```
Shape of DataFrame after removing outliers: (125, 26)
```

In [19]:
```python
# Drop 'CarName' column after extracting company name
car_price.drop('CarName', axis=1, inplace=True)
```

In [21]:
```python
# Calculate correlation matrix
correlation_matrix = car_price.corr()

# Print correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                symboling  fueltype  aspiration  doornumber   carbody  \
symboling        1.000000       NaN   -0.023430    0.618885 -0.590216
fueltype              NaN       NaN         NaN         NaN       NaN
aspiration      -0.023430       NaN    1.000000   -0.026196  0.005757
doornumber       0.618885       NaN   -0.026196    1.000000 -0.667442
carbody         -0.590216       NaN    0.005757   -0.667442  1.000000
drivewheel       0.018977       NaN    0.118590    0.200499 -0.219343
enginelocation        NaN       NaN         NaN         NaN       NaN
wheelbase       -0.478842       NaN    0.285817   -0.358720  0.392167
carlength       -0.262483       NaN    0.279113   -0.335788  0.311228
carwidth        -0.105779       NaN    0.331597   -0.108026  0.100276
carheight       -0.546503       NaN    0.165683   -0.498219  0.589101
curbweight      -0.171182       NaN    0.267033   -0.184789  0.147842
enginetype      -0.000410       NaN   -0.230744    0.052697  0.058624
cylindernumber   0.241794       NaN   -0.054963    0.185567 -0.097062
enginesize      -0.148090       NaN    0.130662   -0.070878  0.010861
fuelsystem       0.151451       NaN    0.223257   -0.005848 -0.057902
boreratio       -0.086965       NaN    0.338007   -0.083835 -0.024504
```

In [22]:
```python
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



Features with relatively high positive correlations with the target variable (price) include: wheelbase, carlength, carwidth, curbweight, enginesize, horsepo
and company. Some features exhibit multicollinearity, such as: carlength, carwidth, and curbweight; enginesize and horsepower.

n [23]:

```
# Selected features
selected_features = ['wheelbase', 'curbweight', 'enginesize', 'horsepower', 'company', 'price']

# Subset the DataFrame with selected features
selected_df = car_price[selected_features]

# Split the data into features (X) and target variable (y)
X = selected_df.drop('price', axis=1)
y = selected_df['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# List of regression algorithms to evaluate
regressors = {
    'Linear Regression': LinearRegression(),
    'Decision Tree Regressor': DecisionTreeRegressor(),
    'Random Forest Regressor': RandomForestRegressor(),
    'Gradient Boosting Regressor': GradientBoostingRegressor(),
    'Support Vector Regressor': SVR()
}

# Train and evaluate each regression algorithm
for name, regressor in regressors.items():
    # Train the model
    regressor.fit(X_train, y_train)

    # Evaluate the model
    train_score = regressor.score(X_train, y_train)
    test_score = regressor.score(X_test, y_test)
```

```
    # Print the evaluation results
    print(f"{name}:")
    print(f"  Training R^2 Score: {train_score:.4f}")
    print(f"  Testing R^2 Score: {test_score:.4f}")
    print("="*50)
```

```
Linear Regression:
  Training R^2 Score: 0.7589
  Testing R^2 Score: 0.7991
==================================================
Decision Tree Regressor:
  Training R^2 Score: 0.9986
  Testing R^2 Score: 0.6753
==================================================
Random Forest Regressor:
  Training R^2 Score: 0.9785
  Testing R^2 Score: 0.8760
==================================================
Gradient Boosting Regressor:
  Training R^2 Score: 0.9933
  Testing R^2 Score: 0.8072
==================================================
Support Vector Regressor:
  Training R^2 Score: -0.1172
  Testing R^2 Score: 0.0019
==================================================
```

==

In this code:

We first specify the selected features ('wheelbase', 'curbweight', 'enginesize', 'horsepower', 'company') and the target variable ('price'). We split the dataset into features (X) and the target variable (y). Then, we split the data into training and testing sets using a test size of 20%. We define a dictionary of regression algorithms to evaluate, including Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, and Support Vector Regressor. We train each regression algorithm on the training data and evaluate its performance using the R^2 score on both the training and testing sets.

Based on the R^2 scores, the Random Forest Regressor seems to perform the best on the testing set, followed by the Gradient Boosting Regressor and Linear Regression. The Decision Tree Regressor overfits the training data as indicated by its high training R^2 score compared to the testing R^2 score. The Support Vector Regressor performs poorly compared to the other models.

5. Classification and Clustering

jupyter 5. Classification and clustering Last Checkpoint: 13 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

Iris Dataset

The iris dataset consists of samples of iris plants, where each sample is characterized by four features: sepal length, sepal width, petal length, and petal width. The target variable represents the species of iris plant for each sample. There are three possible species/classes in the iris dataset:

Setosa Versicolour Virginica. When using machine learning algorithms, the goal is typically to predict the target variable (species of iris) based on the features (sepal and petal measurements). The target variable serves as the ground truth for training and evaluating the models. The target variable is represented numerically in the dataset, where:

0 corresponds to Setosa 1 corresponds to Versicolour 2 corresponds to Virginica

In [1]:
```
from sklearn.datasets import load_iris
#The iris dataset consists of samples of iris plants, where each sample is characterized by four features: sepal length, sepal wi

#Setosa
#Versicolour
#Virginica
```

In [2]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
```

In [3]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [4]:
```
# Step 1: Load the iris dataset
iris_data = load_iris()
X = iris_data.data  # Features
y = iris_data.target  # Target variable
```

In [5]:
```
# Step 2: Explore and understand the dataset

# Print feature names
print("Feature names:", iris_data.feature_names)

# Print target names
print("Target names:", iris_data.target_names)

# Print the shape of the dataset
print("Shape of features:", X.shape)
print("Shape of target:", y.shape)
```

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
Shape of features: (150, 4)
Shape of target: (150,)
```

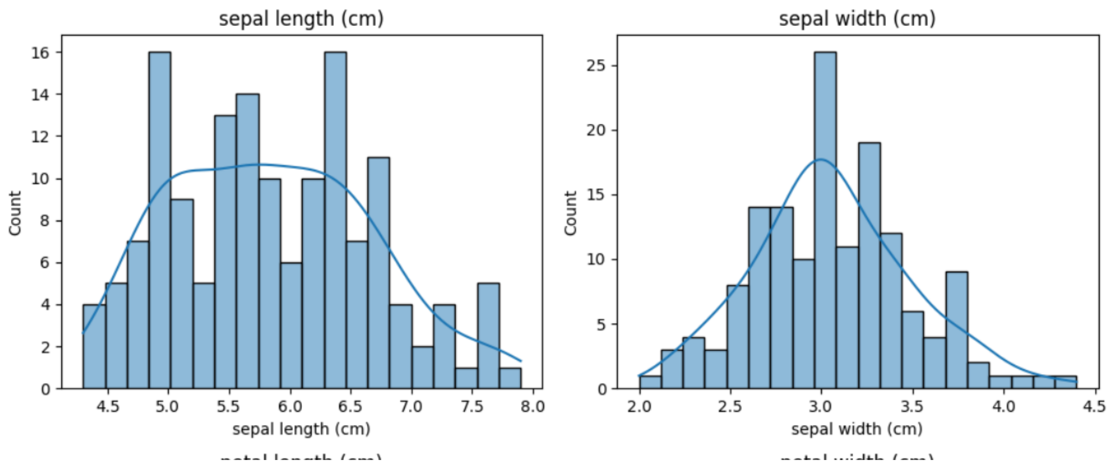
In [13]:
```
# Convert the features and target into a DataFrame for easier exploration
iris_df = pd.DataFrame(data=X, columns=iris_data.feature_names)
iris_df['species'] = y  # Add the target variable to the DataFrame
iris_df['species'] = iris_df['species'].map({0: 'Setosa', 1: 'Versicolour', 2: 'Virginica'})  # Map numerical labels to species r
```

In [14]:
```
# Print descriptive statistics of the features

iris_df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
print("Descriptive statistics of features:")
print(iris_df.describe())
```

```
Descriptive statistics of features:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

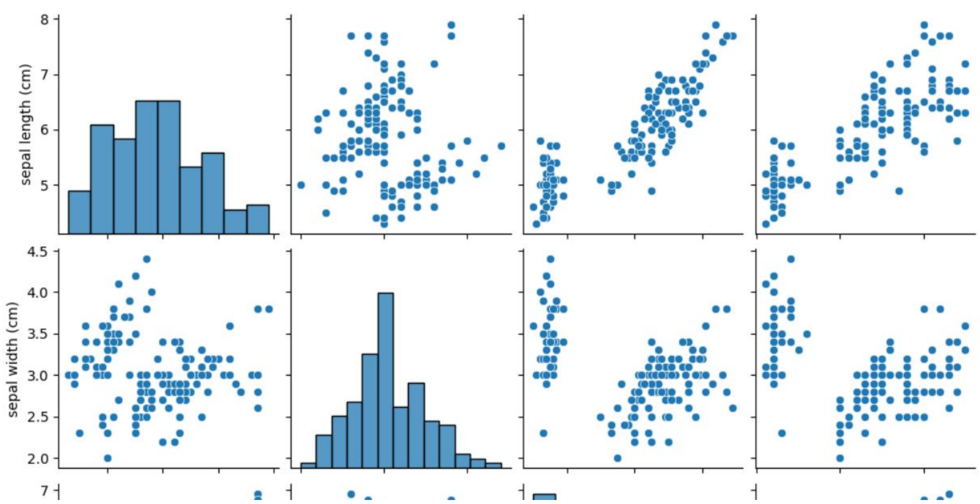
In [15]:

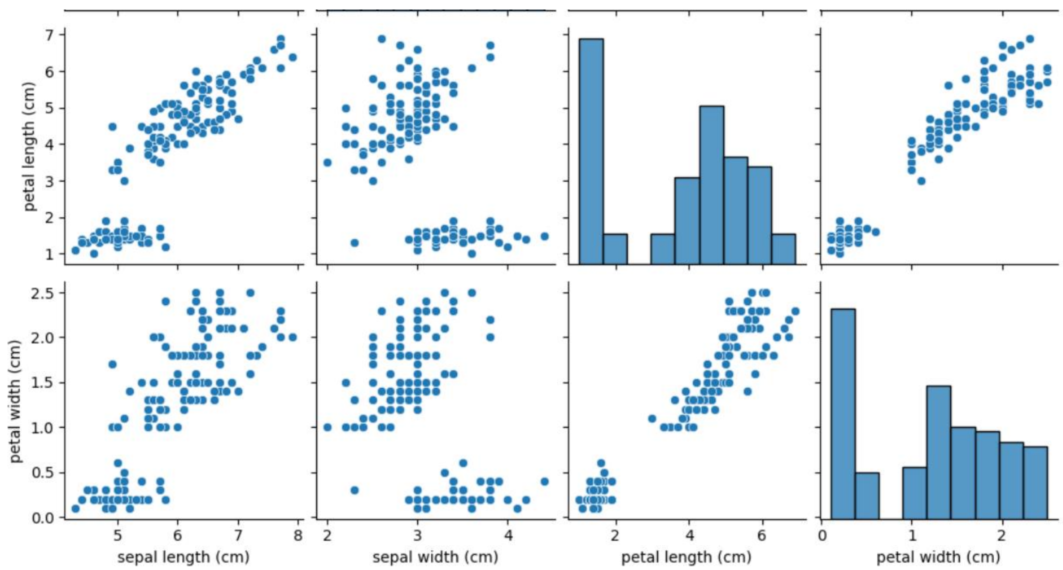
```
plt.figure(figsize=(10, 8))
for i, feature in enumerate(iris_data.feature_names):
    plt.subplot(2, 2, i + 1)
    sns.histplot(iris_df[feature], bins=20, kde=True)
    plt.title(feature)
plt.tight_layout()
plt.show()
```



In [16]:

```
# Visualize the relationship between features using pairplot
sns.pairplot(iris_df)
plt.show()
```





No preprocessing required for this dataset

Logistic Regression

In [17]:
```
# Step 3: Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print the shapes of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (120, 4)
Shape of X_test: (30, 4)
Shape of y_train: (120,)
Shape of y_test: (30,)
```

In [18]:
```
# Initialize the logistic regression model
logistic_reg_model = LogisticRegression(max_iter=1000)

# Train the model on the training data
logistic_reg_model.fit(X_train, y_train)

print("Logistic Regression Model trained successfully!")
```

```
Logistic Regression Model trained successfully!
```

In [19]:
```python
# Step 5: Evaluate Model Accuracy
from sklearn.metrics import accuracy_score

# Make predictions on the testing data
y_pred = logistic_reg_model.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Logistic Regression Model:", accuracy)
```

```
Accuracy of Logistic Regression Model: 1.0
```

In [20]:
```python
# Step 6: Predict New Samples
# Assume 'new_samples' contains features of different samples from the test dataset
new_samples = X_test[:5]  # Example: Predicting the first 5 samples from the test dataset

# Use the trained logistic regression model to predict the classes of new samples
predicted_classes = logistic_reg_model.predict(new_samples)

# Map predicted class indices to species names
predicted_species = [iris_data.target_names[class_index] for class_index in predicted_classes]

# Print the predicted classes
print("Predicted classes of new samples:")
for i, sample in enumerate(new_samples):
    print(f"Sample {i+1}: Features={sample}, Predicted Class={predicted_species[i]}")
```


```python
# Print the predicted classes
print("Predicted classes of new samples:")
for i, sample in enumerate(new_samples):
    print(f"Sample {i+1}: Features={sample}, Predicted Class={predicted_species[i]}")
```


```
Predicted classes of new samples:
Sample 1: Features=[6.1 2.8 4.7 1.2], Predicted Class=versicolor
Sample 2: Features=[5.7 3.8 1.7 0.3], Predicted Class=setosa
Sample 3: Features=[7.7 2.6 6.9 2.3], Predicted Class=virginica
Sample 4: Features=[6.  2.9 4.5 1.5], Predicted Class=versicolor
Sample 5: Features=[6.8 2.8 4.8 1.4], Predicted Class=versicolor
```

Clustering

Clustering

In [21]:
```python
# Step 1: Clustering
from sklearn.cluster import KMeans

# Initialize KMeans with 3 clusters (since there are 3 species)
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit KMeans to the data (excluding target variable)
kmeans.fit(X)

# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
cluster_labels = kmeans.labels_

# Print cluster centers and labels
print("Cluster Centers:")
print(cluster_centers)
print("\nCluster Labels:")
print(cluster_labels)
```

```
Cluster Centers:
[[6.85384615 3.07692308 5.71538462 2.05384615]
 [5.006      3.428      1.462      0.246     ]
 [5.88360656 2.74098361 4.38852459 1.43442623]]

Cluster Labels:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```

In this code:

We initialize a KMeans clustering model with 3 clusters (since there are 3 species of iris flowers). Then, we fit the KMeans model to the data (excluding the target variable). Finally, we print the cluster centers and labels.

Classification

In [22]:

```python
# Step 2: Classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Initialize classifiers
decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
svm_classifier = SVC(kernel='linear', random_state=42)

# Train classifiers on the training data
decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
svm_classifier.fit(X_train, y_train)

# Evaluate classifiers on the testing data
decision_tree_accuracy = decision_tree.score(X_test, y_test)
random_forest_accuracy = random_forest.score(X_test, y_test)
svm_accuracy = svm_classifier.score(X_test, y_test)

# Print accuracy scores
print("Accuracy of Decision Tree Classifier:", decision_tree_accuracy)
print("Accuracy of Random Forest Classifier:", random_forest_accuracy)
print("Accuracy of SVM Classifier:", svm_accuracy)
```

```
Accuracy of Decision Tree Classifier: 1.0
Accuracy of Random Forest Classifier: 1.0
Accuracy of SVM Classifier: 1.0
```


```
Accuracy of Decision Tree Classifier: 1.0
Accuracy of Random Forest Classifier: 1.0
Accuracy of SVM Classifier: 1.0
```


In this code:

We initialize Decision Tree, Random Forest, and Support Vector Machine (SVM) classifiers. Then, we train the classifiers on the training data (X_train, y_train). Next, we evaluate the accuracy of each classifier on the testing data (X_test, y_test) using the score method. Finally, we print the accuracy scores of each classifier.